CERTIFICATE OF TRANSMISSION

I hereby certify that this correspondence (along with any paper referred to as being attached or enclosed) is being submitted *via* the USPTO EFS Filing System on the date shown below to **Mail Stop Appeal Brief-Patents**, Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Date: January 22, 2007 /Jessica Sexton/
Jessica Sexton

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re patent application of:			
Appellant(s):	Christopher L. Anderson, *et al.*	Examiner:	Insun Kang
Serial No:	09/842,527	Art Unit:	2193
Filing Date:	April 25, 2001		

Title: LANGUAGE-NEUTRAL REPRESENTATION OF SOFTWARE CODE ELEMENTS

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Dear Sir:

Appellants' representative submits this brief in connection with an appeal of the above-identified patent application. Payment is being submitted via credit card in connection with all fees due regarding this appeal brief. In the event any additional fees may be due and/or are not covered by the credit card, the Commissioner is authorized to charge such fees to Deposit Account No. 50-1063 [MSFTP194US].

I. Real Party in Interest (37 C.F.R. §41.37(c)(1)(i))

The real party in interest in the present appeal is Microsoft Corporation, the assignee of the present application.

II. Related Appeals and Interferences (37 C.F.R. §41.37(c)(1)(ii))

Appellants, appellants' legal representative, and/or the assignee of the present application are not aware of any appeals or interferences which may be related to, will directly affect, or be directly affected by or have a bearing on the Board's decision in the pending appeal.

III. Status of Claims (37 C.F.R. §41.37(c)(1)(iii))

Claims 1-49 stand rejected by the Examiner. The rejection of claims 1-49 is being appealed.

IV. Status of Amendments (37 C.F.R. §41.37(c)(1)(iv))

No amendments were made to claims after the Final Office Action dated August 24, 2006.

V. Summary of Claimed Subject Matter (37 C.F.R. §41.37(c)(1)(v))

A. Independent Claim 1

Independent claim 1 recites a language-neutral representation of a compile unit transformable to at least one of a plurality of different types of code representations, the language-neutral representation of the compile unit resident on one or more computers and comprising:

a hierarchal arrangement of program elements that neutrally characterize the compile unit; (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

at least one of the program elements representing a type declaration that characterizes at least one class of programmatic constructs of the compile unit; and (*See e.g.*, page 5, line 30 – page 6, line 4)

a code generator that converts the language neutral representation of the compile unit to a corresponding representation of the compile unit in at least one high-level language code. (*See e.g.*, page 6, lines 14-15)

B. Independent Claim 14

Independent claim 14 recites a language-neutral representation of programmatic elements resident upon a computer-readable medium, comprising:

an instance of at least one of a plurality of language-neutral classes, the plurality of classes representing different programmatic constructs of a compile unit and having a hierarchal relationship relative to each other, whereby transformation of the instance into a different representation of the respective programmatic construct is facilitated; and (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

a code generator that converts the instance of at least one of a plurality of language-neutral classes into a corresponding different representation of the instance in at least one high-level language code. (*See e.g.*, page 6, lines 14-15)

C. Independent Claim 23

Independent claim 23 recites a language-neutral representation of computer executable instructions that are transformable to at least one other type of software code representation, the language-neutral representation existent within a computer-readable medium, comprising:

a hierarchal arrangement of objects, each object representing a different program element of the compile unit; (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

at least one class object that represents at least one defined class of program elements of the compile unit; (*See e.g.*, page 5, line 30 – page 6, line 4)

at least one member object associated with the at least one class object that represents computer-executable instructions operable on at least some program elements in the at least one defined class; and (*See e.g.*, page 9, lines 34-35)

a code generator that converts the language-neutral representation of computer executable instructions into a corresponding representation in at least one high-level language code. (*See e.g.*, page 6, lines 14-15)

D. Independent Claim 32

Independent claim 32 recites a language-neutral representation of programmatic elements within a computer-readable medium, comprising:

means for representing different code portions of a compile unit in a language-neutral manner, the means for representing being arranged according to a hierarchy; (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

means for converting the language neutral representation of the compile unit into a representation of the compile unit in a low level language code; and (*See e.g.*, page 6, lines 24-25)

means for converting the language neutral representation of the compile unit into a representation of the compile unit in a high level language code. (*See e.g.*, page 6, lines 14-15)

E. Independent Claim 36

Independent claim 36 recites a computer implemented system that transforms a language-specific representation of a compile unit to a language-neutral representation thereof, the system comprising the following computer executable components:

a plurality of language-neutral classes that represent different programmatic constructs and have a hierarchal relationship relative to each other; (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

an interface that exposes the plurality of classes; (*See e.g.*, page 18, lines 14-17)

a design system that employs the interface to instantiate selected classes of the plurality of classes to create corresponding objects that represent respective code elements of the compile unit arranged according to the hierarchal relationship to define the language-neutral representation; and (*See e.g.*, page 7, lines 11-24)

a code generator that converts the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code. (*See e.g.*, page 6, lines 14-15)

F. Independent Claim 40

Independent claim 40 recites a computer-readable medium having stored thereon computer-executable instructions for creating a language-neutral representation of a compile unit, comprising:

a language-neutral representation comprising associated objects that represent different parts of the compile unit arranged according to an established hierarchy, each of the objects being an instance of a respective class of a plurality of language-neutral classes, each respective class defining a different part of code according to the established hierarchy; and (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

a code generator that converts the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code. (*See e.g.*, page 6, lines 14-15)

G. Independent Claim 41

Independent claim 41 a method to transform a representation of a compile unit in a first high level language code to a language-neutral representation thereof, the method comprising:

mapping each of a plurality of programmatic constructs of the first high level language code to a corresponding class of a plurality of language-neutral classes, the classes having a hierarchal relationship relative to each other; (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

instantiating each corresponding class based on the mapping to create corresponding objects that represent respective programmatic constructs of the compile unit; (*See e.g.*, page 7, lines 11-24)

arranging the objects according to the hierarchal relationship to define the language neutral representation; and (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

converting the language-neutral representation of the compile unit into a corresponding representation in at least one second high-level language code, the second high level language code is in a different high level language than the first high level language code. (*See e.g.*, page 6, lines 6-15)

H. Independent Claim 43

Independent claim 43 recites a computer implemented system to transform a language-neutral representation of a compile unit to a high-level language, comprising the following computer executable components:

an interface that controls manipulation of a plurality of language-neutral program elements of the compile unit, each of the plurality of program elements being an instance of a corresponding class of a plurality of classes arranged according to a hierarchy; and (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-24; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

a code generator that implements the interface to transform each of the program elements to a representation in a high-level language code thereof corresponding to program information associated with each respective program element. (*See e.g.*, page 6, lines 14-15)

I. Independent Claim 47

Independent claim 47 recites a method to transform a language-neutral representation of a compile unit to a corresponding target language-based representation, the language-neutral representation including at least one language-neutral program element, the at least one program element being an instance of a corresponding class of a plurality of classes arranged according to a hierarchy, the method comprising: (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

providing an interface to expose at least one method operative to control at least one of manipulation of program elements and compilation of the language neutral representation; and (*See e.g.*, page 18, lines 14-17)

depending on the target language-based representation, converting the at least one program element to a representation in a high-level language code thereof according to the at least one exposed method or implementing the interface relative to the language neutral

representation to compile each instance of the language-neutral representation into a representation in a low-level language code. (*See e.g.*, page 6, lines 14-25)

J. Independent Claim 48

Independent claim 48 recites a computer implemented system to translate a language-neutral representation of a compile unit to a low-level language, comprising the following computer executable components:

an interface that exposes methods to control compilation of the language-neutral representation; (*See e.g.*, page 6, lines 14-25)

a compiler that implements the interface to translate a plurality of program elements that define the language-neutral representation into the low-level language code, each of the plurality of program elements being an instance of a corresponding class of a plurality of classes that represent program constructs, the plurality of elements being arranged according to a hierarchy; and a code generator that converts the plurality of program elements that define the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code. (*See e.g.*, page 3, lines 19-28; page 5, line 24 – page 6, line 5; page 7, lines 11-16; page 10, lines 30-35; page 15, line 21 – page 16, line 4)

VI. Grounds of Rejection to be Reviewed (37 C.F.R. §41.37I(1)(vi))

A. Whether claims 1-40, 43-46, 48 and 49 are unpatentable under 35 U.S.C. §103(a) over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997).

B. Whether claim 41 and 42 is unpatentable under 35 U.S.C. §103(a) over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997).

C. Whether claim 47 is unpatentable under 35 U.S.C. §103(a) over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997).

VII. Argument (37 C.F.R. §41.37(c)(1)(vii))

A. Rejection of Claims 1-40, 43-46, 48 and 49 Under 35 U.S.C. §103(a)

Claims 1-40, 43-46, 48 and 49 are rejected under 35 U.S.C. §103(a) as being unpatentable over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997). Appellants' representative respectfully requests reversal of this rejection for at least the following reasons. Gustafsson *et al.* in view of Dyer fails to teach or suggest each and every limitation of appellants' claimed invention.

> To reject claims in an application under §103, an examiner must establish a *prima facie* case of obviousness. A *prima facie* case of obviousness is established by a showing of three basic criteria. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) ***must teach or suggest all the claim limitations***. *See* MPEP §706.02(j). The ***teaching or suggestion to make the claimed combination*** and the reasonable expectation of success ***must both be found in the prior art and not based on applicant's disclosure***. *See In re Vaeck,* 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991) (emphasis added).

The subject claims relate to translation between high level, low level and intermediate level language codes in multiple platforms and between computers and devices. This is facilitated by a language-neutral representation as well as interfaces which help convert as per corresponding representation. For instance, the appellants' invention can generate a language neutral representation of a compile unit from a representation of the compile unit in a high level language code, such as C#. The language neutral representation can then be converted to any of a plurality of representations of the compile unit in various high or low level coding languages, such as C, C++, or Assembly. In particular, independent claim 1 (and similarly recited in independent claims 14, 23, 32, 36, 40, 43 and 48) recites ***a language-neutral representation … comprising: a hierarchal arrangement of program elements that neutrally characterize the compile unit … convert the language neutral representation of the compile unit to a corresponding representation of the compile unit in at least one high-level language code.***

Contrary to assertions in the Office Action, Gustafsson *et al.* in view of Dyer, fails to teach or suggest the aforementioned novel aspects of appellants' invention as recited in the subject claims. Gustafsson *et al.* is provided as evidence of prior art disclosing a language neutral representation of a compile unit. However, the cited art discloses compiling a plurality of high level languages into a common runtime representation. Specifically, the common representation is disclosed as the compiled version of one of the high level languages. Gustafsson *et al.* discloses that the compilers for all of the high level languages must be modified to produce output representative of the compiled output of a selected high level language. Therefore, the common representation is not language neutral. Rather, it is specific to one of the high level languages and the compilers for the other high level languages are modified to produce it.

Moreover, Gustafsson *et al*. is directed to incorporating a mixture of languages in a single given software program. This is achieved by enabling the compilers of the languages to allow each other access to their internal data. The congruency of certain languages, which show marked similarities in their syntax, is made use of in the categorization of the data structures. In the portion of the prior art reference cited by the Examiner, the internal data is shared by making use of a common persistent Compiler Symbol Table, wherein all the semantics are stored in an object model. In contrast, the subject claims makes use of the hierarchical arrangement of program constructs to neutrally represent the program elements of the source program. The format options necessary for representation of target program correspond with the hierarchy of the program elements associated with the language-neutral representation. Gustafsson *et al*. fails to disclose "***hierarchal arrangement of program elements that neutrally characterize the compile unit.***"

Furthermore, Dyer is cited as evidence of prior art disclosing converting a language neutral representation into at least one high level language code. However, the cited art discusses examples of Java decompilers. Specifically, conversion of Java class files to Java source code is disclosed. Java class files are not language neutral representations. They are specific to Java source code. As such, Gustafsson *et al*. and Dyer fail to teach a language neutral representation of a compile unit. Therefore, Gustafsson *et al*. and Dyer do not disclose converting a language neutral representation of a compile unit into at least one high level language.

Accordingly, appellants' representative respectfully submits that Gustafsson *et al.* and Dyer, alone or in combination, fail to teach or suggest all limitations of appellants' invention as recited in independent claims 1, 14, 23, 32, 36, 40, 43 and 48 (and claims 1-13, 15-22, 24-31, 33-35, 37-39 and 44-46 that depend respectively therefrom), and thus fails to make obvious the claimed invention. Thus, this rejection should be reversed.

B. Rejection of Claims 41 and 42 Under 35 U.S.C. §103(a)

Claims 41 and 42 are rejected under 35 U.S.C. §103(a) as being unpatentable over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997). Appellants' representative respectfully requests reversal of this rejection for at least the following reasons. Gustafsson *et al.* in view of Dyer fails to teach or suggest each and every limitation of appellants' claimed invention.

Independent claim 41 recites *mapping each of a plurality of programmatic constructs of the first high level language code to a corresponding class of a plurality of language-neutral classes, … arranging the objects … to define the language neutral representation*; ***converting the language-neutral representation of the compile unit into a corresponding representation in at least one second high-level language code, the second high level language code is in a different high level language than the first high level language code***. Gustafsson *et al.* discloses compiling a first high level language code into compiled code representative of a second high level language. However, the cited reference does not suggest converting the language specific compiled code into the second high level language code. Dyer, only discusses a single high level language, Java, and conversion of language specific Java class files into Java source code. As discussed *supra*, Gustafsson *et al.* and Dyer fail to disclose a language neutral representation of a compile unit and also fail to teach or suggest converting a language neutral representation of a compile unit into a high level language code. Therefore, the cited references also do not teach or suggest converting a first high level language code into a language neutral representation and then converting the language neutral representation into a second high level language code that is in a different high level language than the first.

In view of the foregoing, it is readily apparent that the cited references either alone or in combination do not teach or suggest Appellants' invention as recited in independent claim 41

and (and all claim42 which respectively depends there from). Therefore, it is respectfully requested that this rejection should be reversed.

C. Rejection of Claims 47 Under 35 U.S.C. §103(a)

Claim 47 is rejected under 35 U.S.C. §103(a) as being unpatentable over Gustafsson *et al.* (US 6,067,413) in view of Dyer ("Java Decompiles compared," Java World, 7/1997). Appellants' representative respectfully requests reversal of this rejection for at least the following reasons. Gustafsson *et al.* in view of Dyer fails to teach or suggest each and every limitation of appellants' claimed invention.

Independent claim 47 recites a method to ***the language-neutral representation including at least one language-neutral program element, the at least one program element being an instance of a corresponding class of a plurality of classes arranged according to a hierarchy… converting the at least one program element to a representation in a high-level language code thereof according to the at least one exposed method***. As noted above with respect to the similarly recited limitations of independent claims 1, 14, 23, 32, 36, 40, 43 and 48, Gustafsson *et al.* and Dyer fail to disclose a language neutral representation of a compile unit and also fail to teach or suggest converting a language neutral representation of a compile unit into a high level language code.

Thus, it is readily apparent that the cited references either alone or in combination do not teach or suggest Appellants' invention as recited in independent claim 47. Accordingly, it is respectfully requested that this rejection should be reversed.

CONCLUSION

The present application is believed to be in condition for allowance in view of the above comments. A prompt action to such end is earnestly solicited.

In the event any fees are due in connection with this document, the Commissioner is authorized to charge those fees to Deposit Account No. 50-1063 [MSFTP194US].

Should the Examiner believe a telephone interview would be helpful to expedite favorable prosecution, the Examiner is invited to contact appellants' undersigned representative at the telephone number below.

Respectfully submitted,

AMIN, TUROCY & CALVIN, LLP

/Himanshu S. Amin/
Himanshu S. Amin
Reg. No. 40,894

AMIN, TUROCY & CALVIN, LLP
24TH Floor, National City Center
1900 E. 9TH Street
Cleveland, Ohio 44114
Telephone (216) 696-8730
Facsimile (216) 696-8731

VIII. Claims Appendix (37 C.F.R. §41.37(c)(1)(viii))

1. A language-neutral representation of a compile unit transformable to at least one of a plurality of different types of code representations, the language-neutral representation of the compile unit resident on one or more computers and comprising:

a hierarchal arrangement of program elements that neutrally characterize the compile unit;

at least one of the program elements representing a type declaration that characterizes at least one class of programmatic constructs of the compile unit; and

a code generator that converts the language neutral representation of the compile unit to a corresponding representation of the compile unit in at least one high-level language code.

2. The language-neutral representation of claim 1, further comprising a collection of at least one member that characterizes programmatic attributes associated with and able to be implemented within the at least one class.

3. The language-neutral representation of claim 2, wherein the collection further comprises an expression class within the at least one class.

4. The language-neutral representation of claim 2, wherein the collection further comprises a statement class within the at least one class.

5. The language-neutral representation of claim 2, wherein the hierarchal arrangement further comprises a namespace that contains the at least one class.

6. The language-neutral representation of claim 1, wherein at least one of the program elements of the hierarchal arrangement encapsulates another of the program elements.

7. The language-neutral representation of claim 1 in combination with an interface associated with the language-neutral representation, the interface transforms the language-neutral representation to a corresponding desired code representation.

8. The language-neutral representation of claim 7, wherein the program elements comprise objects, each object exposing at least one of a method, attribute, and property of each respective object, the interface employs the at least one of method, attribute and property to facilitate the transformation into the desired code representation.

9. The language-neutral representation of claim 7, wherein the interface further comprises a compiler interface programmed to transform the language-neutral representation to a corresponding representation in a low-level language code.

10. The language-neutral representation of claim 9, wherein the representation in a low-level language code further comprises an assembly of computer-executable instructions.

11. The language-neutral representation of claim 7, wherein the interface further comprises a code generator interface programmed to convert the language-neutral representation to a plurality of corresponding representations, wherein each representation is in a different high-level language code.

12. The language-neutral representation of claim 1, wherein the program elements comprise instances of a plurality of language-neutral classes, each instance defining an associated object.

13. The language-neutral representation of claim 12, wherein at least one associated object represents the type declaration, at least another object being encapsulated within the at least one object representing the at least one type declaration, the at least another object representing program code of the compile unit that derives from a class associated with the at least type declaration.

14. A language-neutral representation of programmatic elements resident upon a computer-readable medium, comprising:

an instance of at least one of a plurality of language-neutral classes, the plurality of classes representing different programmatic constructs of a compile unit and having a hierarchal

relationship relative to each other, whereby transformation of the instance into a different representation of the respective programmatic construct is facilitated; and

a code generator that converts the instance of at least one of a plurality of language-neutral classes into a corresponding different representation of the instance in at least one high-level language code.

15. The language-neutral representation of claim 14, further comprising a plurality of instances of the plurality of the classes, wherein each instance of a corresponding class of the plurality of classes represents a respective programmatic construct of the compile unit, the plurality of instances being organized in a hierarchal relationship based on the classes associated with the plurality of instances and relationships among the programmatic constructs represented thereby.

16. The language-neutral representation of claim 15, wherein each of the plurality of instances exposes at least one item associated with the programmatic construct represented thereby.

17. The language-neutral representation of claim 16, wherein at least one of the plurality of instances represents a type declaration, at least another instance being encapsulated within the instance representing the type declaration, the at least another instance representing a programmatic construct that derives from the at least type declaration.

18. The language-neutral representation of claim 17 wherein the at least another object further comprises at least one of a statement and an expression.

19. The language-neutral representation of claim 16 in combination with an interface that transforms the language neutral representation to the different representation, the interface employs the at least one item to facilitate the transformation of the language-neutral representation into the different representation.

20. The language-neutral representation of claim 19, wherein the interface further comprises a compiler interface programmed to transform the language-neutral representation to the corresponding different representation in a low-level language code.

21. The language-neutral representation of claim 20, wherein the different representation in a low-level language code further comprises an assembly of computer-executable instructions.

22. The language-neutral representation of claim 19, wherein the interface further comprises a code generator interface programmed to generate a plurality of corresponding representations from the language-neutral representation, wherein each representation is in a different high-level language code from the language-neutral representation.

23. A language-neutral representation of computer executable instructions that are transformable to at least one other type of software code representation, the language-neutral representation existent within a computer-readable medium, comprising:

a hierarchal arrangement of objects, each object representing a different program element of the compile unit;

at least one class object that represents at least one defined class of program elements of the compile unit;

at least one member object associated with the at least one class object that represents computer-executable instructions operable on at least some program elements in the at least one defined class; and

a code generator that converts the language-neutral representation of computer executable instructions into a corresponding representation in at least one high-level language code.

24. The language-neutral representation of claim 23, further comprising a namespace object that represents a namespace of the compile unit, the namespace object comprising a collection of class objects including the at least one class object.

25. The language-neutral representation of claim 24, further comprising a plurality of member objects associated with the at least one class object, wherein the at least one class object represents a common base class that is shared by the plurality of member objects.

26. The language-neutral representation of claim 24, wherein the plurality of member objects further comprise a collection of objects representing at least one of a statement and an expression of the compile unit.

27. The language-neutral representation of claim 23 in combination with an associated interface, the interface transforms the language-neutral representation to a corresponding desired code representation.

28. The language-neutral representation of claim 27, wherein each object exposes at least one item indicative of the program element represented thereby, the interface employs each exposed item to facilitate transformation of the language-neutral representation into the desired code representation.

29. The language-neutral representation of claim 28, wherein the interface further comprises a compiler interface that exposes computer-executable instructions to transform the language-neutral representation to a corresponding representation in a low-level language code.

30. The language-neutral representation of claim 29, wherein the representation in a low-level language code further comprises an assembly of computer-executable instructions.

31. The language-neutral representation of claim 28, wherein the interface further comprises a code generator interface that exposes computer-executable instructions to convert the language-neutral representation to a plurality of corresponding representations, wherein each representation is in a different high-level language code.

32. A language-neutral representation of programmatic elements within a computer-readable medium, comprising:

means for representing different code portions of a compile unit in a language-neutral manner, the means for representing being arranged according to a hierarchy;

means for converting the language neutral representation of the compile unit into a representation of the compile unit in a low level language code; and

means for converting the language neutral representation of the compile unit into a representation of the compile unit in a high level language code.

33. The language-neutral representation of claim 32, further comprising means for transforming the language-neutral representation to a plurality of corresponding desired representations of code.

34. The combination of claim 33, wherein the means for transforming further comprises means for compiling the language-neutral representation to a plurality of corresponding representations, wherein each representation is in a different low-level language code representation.

35. The language-neutral representation of claim 33, wherein the means for transforming further comprises means for generating a plurality of corresponding representations from the language-neutral representation, wherein each representation is in a different high-level language code.

36. A computer implemented system that transforms a language-specific representation of a compile unit to a language-neutral representation thereof, the system comprising the following computer executable components:

a plurality of language-neutral classes that represent different programmatic constructs and have a hierarchal relationship relative to each other;

an interface that exposes the plurality of classes;

a design system that employs the interface to instantiate selected classes of the plurality of classes to create corresponding objects that represent respective code elements of the compile unit arranged according to the hierarchal relationship to define the language-neutral representation; and

a code generator that converts the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code.

37. system of claim 36, wherein at least one of the objects is a namespace object that comprises a collection of at least one base class member object that provides a common base class for at least some of the objects.

38. The system of claim 37, wherein the at least one base class further comprises at least one member comprising a collection of objects representing at least one of a statement and an expression of the compile unit.

39. The system of claim 36, wherein the code generator interface is programmed to transform the language-neutral representation to a high-level language code that is different from the language-specific representation.

40. A computer-readable medium having stored thereon computer-executable instructions for creating a language-neutral representation of a compile unit, comprising:

a language-neutral representation comprising associated objects that represent different parts of the compile unit arranged according to an established hierarchy, each of the objects being an instance of a respective class of a plurality of language-neutral classes, each respective class defining a different part of code according to the established hierarchy; and

a code generator that converts the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code.

41. A method to transform a representation of a compile unit in a first high level language code to a language-neutral representation thereof, the method comprising:

mapping each of a plurality of programmatic constructs of the first high level language code to a corresponding class of a plurality of language-neutral classes, the classes having a hierarchal relationship relative to each other;

instantiating each corresponding class based on the mapping to create corresponding objects that represent respective programmatic constructs of the compile unit;

arranging the objects according to the hierarchal relationship to define the language neutral representation; and

converting the language-neutral representation of the compile unit into a corresponding representation in at least one second high-level language code, the second high level language code is in a different high level language than the first high level language code.

42. The method of claim 41, further comprising transforming the language-neutral representation into a corresponding representation of the compile unit in a high-level language code that is the same as the representation of the compile unit in the first high level language code.

43. A computer implemented system to transform a language-neutral representation of a compile unit to a high-level language, comprising the following computer executable components:

an interface that controls manipulation of a plurality of language-neutral program elements of the compile unit, each of the plurality of program elements being an instance of a corresponding class of a plurality of classes arranged according to a hierarchy; and

a code generator that implements the interface to transform each of the program elements to a representation in a high-level language code thereof corresponding to program information associated with each respective program element.

44. The system of claim 43, wherein the interface further comprises a class interface to expose program information relating to a base class of the compile unit representing at least one type declaration.

45. The system of claim 44, wherein the interface further comprises a member interface that exposes program information relating to a member of the base class of the compile unit.

46. The system of claim 43, wherein the member of the base class represents a base class for at least one of a statement and an expression.

47. A method to transform a language-neutral representation of a compile unit to a corresponding target language-based representation, the language-neutral representation including at least one language-neutral program element, the at least one program element being an instance of a corresponding class of a plurality of classes arranged according to a hierarchy, the method comprising:

providing an interface to expose at least one method operative to control at least one of manipulation of program elements and compilation of the language neutral representation; and

depending on the target language-based representation, converting the at least one program element to a representation in a high-level language code thereof according to the at least one exposed method or implementing the interface relative to the language neutral representation to compile each instance of the language-neutral representation into a representation in a low-level language code.

48. A computer implemented system to translate a language-neutral representation of a compile unit to a low-level language, comprising the following computer executable components:

an interface that exposes methods to control compilation of the language-neutral representation;

a compiler that implements the interface to translate a plurality of program elements that define the language-neutral representation into the low-level language code, each of the plurality of program elements being an instance of a corresponding class of a plurality of classes that represent program constructs, the plurality of elements being arranged according to a hierarchy; and a code generator that converts the plurality of program elements that define the language-neutral representation of the compile unit into a corresponding representation in at least one high-level language code.

49. The system of claim 48, wherein the low-level language code comprises at least one of an assembly, byte code and intermediate language.

IX. Evidence Appendix (37 C.F.R. §41.37(c)(1)(ix))

None.

X. Related Proceedings Appendix (37 C.F.R. §41.37(c)(1)(x))

None.